CSE 143 Final Exam
Part 1 - August 18, 2011, 9:40 am

Name _____ Student ID # _____

Section _____ TA Name _____

The exam is closed book, closed notes, closed devices, except that you may have a 5x8 card with hand-written notes plus the reference sheet handed out with the exam. There are two blank pages at the end of the exam if you need extra scratch space to write.

You must show your ID and hand in your exam as you leave the room. You may not come back inside after that.

Style and indenting matter, within limits. We're not overly picky about details like an extra or a missing parenthesis, but we do need to be able to follow your code and understand it. A few well-chosen comments can be very helpful. Too many make it hard to read the code.

If you have questions during the exam, raise your hand and someone will come to you. *Don't* leave your seat.

Please wait to turn the page until everyone has their exam and you have been told to begin.

Advice: The solutions to many of the problems are quite short. Don't be alarmed if there is a lot more room on the page than you actually need for your answer.

More gratuitous advice: Be sure to get to all the questions. If you find you are spending a lot of time on a question, move on and try other ones, then come back to the question that was taking the time.

1 – Trees	/ 16
2 – Trees	/ 8
3 – BST search	/ 20
4 – Collections	/ 20
5 – Class design	/ 16
6 – Types	/ 10
Total	/ 90

Question 1. (16 points) Binary Search Trees. (a) Draw a picture that shows the integer binary search tree that results when the following numbers are inserted into a new, empty binary search tree in the order given:

42 17 10 50 15 20 12

(b) Write down the order in which the nodes in your tree would be visited using the following tree traversal orders:

preorder _____

postorder _____

inorder _____

Question 2. (8 points) Binary Search Trees. Draw a picture that shows an integer binary search *with **minimum height*** that contains the following numbers. If there is more than one possible way to store the nodes in a binary search tree with minimum height, then you can show any of the possible solutions.

42 17 10 50 15 20 12

(Note: This is exactly the same data as in the previous problem, except that this time you're being asked to construct a minimum height binary search tree that contains the numbers. The numbers can be added to the tree in any order, not necessarily the one given. Hint: in a minimum-height tree, both subtrees of each interior node will have approximately the same number of nodes.)

Question 3. (20 points) Tree search. Suppose we have a class that stores a set of integer values with no duplicates. The class uses a binary search tree internally to store the values.

For this problem, complete the definition of method `numLessThan` below so it returns the number of items in the set whose value is less than the method's argument `n`. You may define additional auxiliary methods if you find them useful. The next page is blank to provide additional space for your answer if needed. You may define as many additional simple variables as you wish, but no additional collections, arrays, or tree nodes, and you may not make any changes to the tree itself.

For full credit, your solution must take advantage of the binary search tree organization and not examine any more of the tree than necessary.

```
public class IntTreeSet {
   // tree nodes
   private class IntTreeNode {
      public int val;            // data value in this node
      public IntTreeNode left;   // left subtree with values < val
      public IntTreeNode right;  // right subtree with values > val
   }

   // binary search tree storing contents of this IntTreeSet
   private IntTreeNode treeRoot;

   // construct new empty IntTreeSet
   public IntTreeSet() {
      treeRoot = null;
   }

   // code for other methods omitted to save space...

   // return number of items in this set with value < n
   public int numLessThan(int n) {

      // write your answer below and on the next page if needed.









   }
```

Question 3. (cont.) Additional space for answer if needed.

Question 4. (20 points) Collections. Complete the following method so it will scan a list of words, find all of the words in that list that do not appear in a dictionary, and record how many times those words not in the dictionary appear in the original list.

The input parameters to the method are a List (words) of strings and a Set (dictionary) of strings. The method should return a Map containing <String,Integer> pairs, where the String in each pair is a word that appears in the input list but not in the dictionary, and the associated integer is the number of times the corresponding word appeared in the list. The entries in the resulting Map do not need to be sorted in any particular order. You may create additional collections if you need them. String values must match exactly to be considered to be the same: i.e., "foo", "Foo", "FOO" and " FOO " are all different.

```
// Return a Map containing a list of <String,Integer> pairs where
// each String is a String that appears in the List words but not
// in the Set dictionary, and the associated Integer is the number
// of times the word appears in the List words

Map<String,Integer> countUnknownWords(List<String> words,
                                      Set<String> dictionary) {









}
```

Question 5. (16 points) Class design. For an address book application we have created the following class to represent the name and address of each entry in the address book.

```
public class AddressCard {
   // instance data
   private String firstName;
   private String lastName;
   private String city;
   private int zipCode;

   // Create new AddressCard with given data
   public AddressCard(String fn, String ln, String city, int zip) {
      firstName = fn;  lastName = ln; this.city = city; zipCode = zip;
   }

   // add any additional code needed below









}
```

We would like to modify this class so we can compare one `AddressCard` object to another. If `x` and `y` are `AddressCard` objects, then `x.compareTo(y)` should compare `x` to `y` and return an appropriate integer depending on the ordering of `x` and `y`. `AddressCards` are ordered by their `lastName` fields. If two cards have the same `lastName`s, then the `firstName`s are used to decide the ordering. The `lastName` and `firstName` strings should be compared using their natural ordering as Strings and should not be modified. In particular, two strings that are similar but differ only in capitalization ("Abc" and "abc") are not equal.

Make the necessary modifications and additions to the above class to allow `AddressCard` objects to be compared.

Question 6. (10 points) Abstract classes and interfaces. (Brief, to-the-point answers are appreciated)

In addition to regular classes, both abstract classes and interfaces can be used in Java programs to define new types.

(a) Describe one advantage of defining a new type using an abstract class instead of an interface.

(b) Describe one advantage of defining a new type using an interface instead of an abstract class.

Additional blank page for scratch paper.

If you write an answer to a test question here, be sure to go back to the original page and write a note so the grader can find your answer.

Additional blank page for scratch paper.

If you write an answer to a test question here, be sure to go back to the original page and write a note so the grader can find your answer.